(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 464 715 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
28.01.1998 Bulletin 1998/05

(51) Int Cl.[6]: **G06F 9/46**

(21) Application number: **91110772.0**

(22) Date of filing: **28.06.1991**

(54) Interlock queueing

Warteschlangen für gegenseitige Verriegelung

Mise en file d'attente d'entrelacement

(84) Designated Contracting States:
DE FR GB IT NL

(30) Priority: **29.06.1990 US 546365**

(43) Date of publication of application:
08.01.1992 Bulletin 1992/02

(73) Proprietor: **DIGITAL EQUIPMENT CORPORATION**
Maynard Massachusetts 01754-1418 (US)

(72) Inventor: **Ramanujan, Raj**
Leominster, Massachusetts 01453 (US)

(74) Representative: **Betten & Resch**
Reichenbachstrasse 19
80469 München (DE)

(56) References cited:

- **INFORMATION SYSTEMS vol. 14, no. 2, 1989, DORTMUND, GERMANY pages 175 - 180; KELTER: 'Deadlock-freedom of large transactions in object management systems'**
- **PERFORMANCE EVALUATION REVIEW vol. 17, no. 1, May 1989, BERKELY, USA pages 49 - 60; ANDERSON ET AL: 'The performance implications of thread management alternatives for shared-memory multiprocessors.'**

EP 0 464 715 B1

Printed by Jouve, 75001 PARIS (FR)

Description

Field Of The Invention

This invention relates to a computer system and, more particularly, to a lock out avoidance mechanism used in the computer system to prevent a situation wherein lock requests generated by the system devices are never serviced by the system.

Background of The Invention

In multiprocessor systems, any of a number of processors or devices may desire exclusive access to a shared resource in the system. These shared resources can be, for example, a memory location such as a block of memory, longword, byte, etc. In order to obtain exclusive rights to the memory location, the processor generates an interlock signal in the form of a lock request which will lock that particular location in memory to a device requesting access and thus deny any other processor or device interlocked access to the particular memory location. Once the processor finishes with the particular memory location, an unlock request is generated to unlock the location, thus again allowing general access.

The use of lock requests occurs most often when processors share a resource such as a particular data structure or table stored in the memory. In order to gain access to such shared resource, the processors synchronize using interlocked operation in memory locations sometimes referred to as semaphores. Because only one processor is granted exclusive access to the semaphore any operations on the shared resource is thus serialized. It is possible that multiple processors wishing to access the resource simultaneously may generate lock requests for the semaphore. For example, if a system includes four processors (CPUs) or nodes from which lock requests can originate, it is possible for two or more processors to desire interlocked access to the same location at the same time. In this instance, one of the requests is refused and the refused node receives a retry signal instructing it to retry the lock request. Typically, each of the CPUs or the memory itself must keep track of the locked memory locations, status of any pending lock requests, etc. In other systems using a central arbiter to receive the lock requests, the tracking of the lock requests occurs at the central arbiter. In either system it is possible, when multiple nodes require locked access to the same memory location, for one or more nodes to be denied access forever. This condition is referred to as lockout.

Further, many systems include several levels of processing, with each level having multiple sources which can generate lock requests. Therefore, lock out may occur at any level in the system wherein multiple lock requests are generated. In previous system designs the lock out condition was resolved either by throttling all requestors except the node locked out when this condition was detected by the system or by forcing queuing of all locked requests which enforced sequentiality. This resulted in either a complex lockout avoidance protocol or slowed the system performance with respect to lock requests.

Summary of the Invention

Prior art publication by Kelter in Information Systems, vol. 14, no. 2, 1989, Dortmund, Germany, pages 175 to 180, titled "Deadlock-Freedom of Large Transactions in Object Management Systems", teaches how to prevent deadlocks between large transactions in object management systems. In object management systems, deadlocks cause undesirable consequences and should be avoided. To this end, a technique called pseudo preclaiming is used. A queue protocol taught in the prior art publication allows granting of locks without delay on unlocked objects. Pseudo preclaiming protocol taught by this prior art arrangement guarantees that transactions which do not claim further objects will not run into deadlocks.

The invention, in its broad form, resides in a method for avoiding lockout of lock requests from a plurality of nodes to share a common resource, and a lockout avoidance circuit, as recited respectively in claims 1 and 5.

The present invention overcomes the prior art problems by providing an efficient and simple hierarchical system of lock out avoidance mechanisms for use in systems having central locations for receiving the lock requests. To avoid lock out, each of the mechanisms uses a centralized queuing structure called a lock queue. The centralized queuing mechanism is used because all interlock signals, e.g., lock requests and unlock requests, are received at this central location. The lock queue contains as many entries as there are possible nodes from which lock requests can originate, but no one entry is specifically tied to any node. Each entry in the lock queue contains a valid indication field and a node identification (I.D.) field. The valid indication field indicates whether the lock request is valid and the node I.D. field stores the identification of the node generating the lock request. The lock queue further has a head and a tail entry. The head points to the first entry, i.e., the highest priority entry, in the queue and the tail points to the first available entry in the lock queue. Each time the head of the queue is serviced, i.e., its lock request is satisfied, the head of the queue moves to the next entry in line. If one of the nodes already has an entry in the queue which has not yet been serviced, then subsequent lock requests from that node will be written into the same place in the lock queue. This is possible because the entries only contain the node I.D. and not the request itself. In this manner, the lock queue need only contain a number of entries corresponding to the number of nodes from which lock requests can origi-

nate.

The lock queue is usually empty and lock requests, from the various nodes passing through the central location, only enter the lock queue, if, after a predetermined number of tries, they are unable to gain access to a particular resource. For example, assume that a lock request from node 0 is refused access because the particular memory location was already locked or a register for allowing the lock request was not available, then the central location increments a refusal counter for the particular node. When the node, in this case node 0, retries its lock request, it will then again be either refused or granted access. If the lock request is refused a predetermined consecutive number of times while the lock queue is empty, then the node's I.D. will be stored at the head of the lock queue along with an indication of its validity.

Because the lock queue is no longer empty, any subsequent lock requests from any other node in the system will be refused access to any location and will immediately enter the lock queue in the order in which they are serviced by the central location. When a lock request at the head of the queue is serviced, then the next waiting request in the queue becomes the head of the queue.

In this manner, a lock out avoidance mechanism is operated such that if the lock queue is empty, it is difficult to enter the lock queue because a particular lock request must be refused the predetermined number of times before the lock request may enter the queue. Once in the queue, the lock requests are serviced serially which slows down the system. Therefore, the lock queue is operated to try to keep the queue empty by using the refusal counter. This operation ensures that the queue is usually empty. However, once the lock queue is no longer empty, all future lock requests are serviced strictly by their queue position. This operation continues until the lock queue again becomes empty. Thus, the lock out avoidance mechanism at this level guarantees that each node will eventually have its lock request serviced.

The present invention provides a similar lock out avoidance scheme at each level in the system wherein a lock out condition can occur. For example, a typical multiprocessor system has a number of hierarchical levels. A first level has each of the processors in the system vying for access to the system bus. The second level has devices within each processor, such as a processor chip and an interface chip, vying for access to the processor's bus. Further levels can include devices coupling through the interface chip which try to access the interface bus, etc.

Assuming in our second level that for each processor of a multiprocessor system, there are two sources of lock requests, i.e., the processor chip itself and other devices interfacing therewith, then the lock out avoidance mechanism at this level consists of a two entry queue. Each entry in the queue contains two fields: 1) an entry valid and 2) source I.D. As long as the lock requests are not refused, the queue will remain empty. However, if a lock request from, for example, the processor chip is refused, then that lock request immediately enters the head of the queue. From that point onward until the request is granted, only the processor chip's lock requests can be output onto a bus to the central location. If another device generates a lock request at the second level, the lock request enters the queue behind the processor chip's entry and is internally refused by the processor. When the head of the queue's lock request is eventually satisfied, i.e., the processor chip is finally granted access, the entry is popped from the queue and the next device becomes the head of the queue. Thus, a lock out avoidance mechanism is provided at a second level in the system.

As a result of providing a lock out avoidance mechanism at each level wherein multiple sources can generate lock requests and hence lock outs could occur, the present invention guarantees that every lock request generated in the system will eventually be serviced.

Brief Description Of The Drawings

Figure 1 is a block diagram of a system advantageously employing the present invention.

Figure 2 is a block diagram of the central arbiter including the lock logic of the present invention.

Figure 3 is a block diagram of one of the CPU modules in Figure 1.

Figure 4 is a detailed block diagram of the lock logic of the present invention.

Figure 4a is a block diagram of a lock register entry.

Figure 4b is a diagram of an entry in the lock queue.

Figure 5 is a diagram of the lock queue at the processor level in the system of Figure 1.

Detailed Description

System Overview

Fig. 1 is a block diagram of a system which advantageously uses the present invention. As illustrated, there are four CPU's 0-3 indicated as blocks 11-14, respectively. These blocks 11-14 are coupled to a central unit, indicated by dashed line 15, to be described in more detail below. Each of the CPUs 11-14 is connected to the central unit 15 over a point to point bus, hereinafter referred to as an E-BUS TA, the E-BUS TA for CPU 0 being designated as 17, the corresponding bus for CPU 1 as 18, and so on accordingly. Each CPU 11-14 also receives output on an E-BUS FA from the central unit 15, the four buses being designated respectively as 21 through 24. Also shown in the illustrated embodiment, each of the CPUs 11-14 communicates with a bus adaptor XJA 25 over two 16-bit buses, i.e., an input and an output bus, with the bus adaptor 25 converting this information to another bus 27. A console 26 is associated with at least one of the CPUs 11-14.

The system also includes a shared memory made up of a plurality of memory modules 31a to 31n which are coupled to the central unit 15 over what is designated A-BUS FA 33 and what is designated as A-BUS TA 35. The A-BUS FA 33 carries outputs from the central unit 15 to the memory modules and the bus 35 provides memory information to the central unit 15. In the illustrated embodiment, buses 17-24 and 33 are 32-bit parallel buses and bus 35 is a 64-bit parallel bus. The E-BUSES operate at 8ns timing and the A-BUSES at 16ns in this embodiment.

Fig. 1 further shows a configuration for the central unit 15. A multiplexer, designated generally by 37, is provided in the central unit 15. As will be shown below, the multiplexer 37 is made up of several multiplexers advantageously coupled together. At the inputs to the multiplexer 37 is a port logic block 49 composed of port logic units 49a. Each port logic unit 49a includes a buffer 53 for each input port, i.e., each E-BUS TA 17-20. Each of the buffers 53 in the port logic units 49a can hold three words for each of buses 17-20 from CPUs 11-14. Outputs from the buffers 53 are coupled into scheduler arbitrator logic 50 which arbitration logic 50 provides outputs to control multiplexer 37. One of the general outputs of MUX 37 is fed to a write buffer 10 which is coupled to memory modules 31a-n via the bus 33.

Fig. 2 is a more detailed block diagram of the central unit 15 of Fig. 1 showing a port select logic 65 and resource check 67 which together compose the scheduler/arbitrator 50 of Fig. 1 and its relationship to other elements of the system. Port select logic 65 is combined with multiplexer 37a to form scheduling logic 66. Resource check logic 67 combines with MUX 37b and state device 42 to form arbitrator 51. As illustrated, the buses 17-20 are inputs to the port logic 49. Each CPU is considered a system port. Individual port logic units, designated 49a-d, are provided for each of the buses 17-20. The bus information enters a state device 53a, forming part of the buffer 53 of Fig. 1, from which it is directed to buffer 53b, also forming part of buffer 53 in Fig. 1. The buffer 53b has storage space for three words. The output from the state device 53a is also provided into validity logic 57. The validity logic 57 controls a port multiplexer 59. The validity logic 57 also receives a port grant signal on line 61 from the arbitrator 51. Validity logic 57 determines whether or not a command or data are valid and which of the data, either in the buffers 53b or on the input line from the state device 53a, is to be switched out onto an output line 63 which provides one of the inputs to a multiplexer 37a (part of MUX 37 in Fig. 1). The outputs on lines 63 from the port multiplexers 59 are the inputs to the multiplexer 37a. Associated with the multiplexer 37a is port selection or scheduling logic 65. In response to outputs from the arbitrator 51, and its own logic, the port select logic 65 selects one of the four input lines 63 to be coupled to the output of multiplexer 37a. This, of course, must be coordinated with the operation of the logic 57 which is switching outputs onto the lines 63. The port select logic 65 basically operates so as to provide the four ports providing their inputs on line 63 a round robin access to bus 36.

Bus 36 is an input to a resource check block 67 which is part of the arbitrator 51. Bus 36 is also an input to the multiplexer 37b (also part of MUX 37 in Fig. 1). Bus 36 is also coupled to a memory map unit ("MMAP") 69, lock logic unit ("LOCK") 71, input/output unit ("CPIO") 73, interrupt request unit ("IREQ/SNIT") 75, memory controller ("MEMC/DBEC") 77 and memory write data path unit ("MWDP") 79. Each of the blocks 69, 71, 73. 75 and 77 also provide inputs to the multiplexer 37b. In addition, the resource check 67 can provide an ARB command to the multiplexer 37b.

The previously described A-BUS TA is provided as an input to a memory read data path 81 which provides its input into the memory controller, i.e., the MEMC/DEBC 77, the output of which is the memory controller refill data which is one of the inputs to multiplexer 37b. An output of MEMC/DBEC 77 is also provided onto the line 33, i.e., the A-BUS FA line. The output from the multiplexer 37b is provided through a state device 42. The output of state device 42 forms the E-BUS FA buses that couple to all the CPUs.

Each of the blocks 69, 71, 73, 75, 77 and 79 provide an input to the resource check 67 which utilizes this information and the presence of a command on the bus 36 to arbitrate between the different inputs which want access to the multiplexer 37b. It grants access via its output lines coupled through state devices 67a and 67b. One set of output lines is the output lines 83 leading from the resource check 67 to each of the blocks 69, 71, 73, 75, 77 and 79. The other are the lines 61 and 85 leading, respectively, to the port logic 49 and the port select logic 65.

Fig. 3 is a more detailed block diagram of one of the CPU modules 11-14 (in this case CPU 0) shown in Fig. 1. The CPU module 11 has three major component chips including an interface (X) chip 102, a processor (P) chip 104 and cache interface (CF) chip 100. The processing chip or P chip 104 is coupled with a cache memory 106. The processing chip 104 further couples with a cache tag store 110 by which it requests an address over line 112 and receives a tag entry via line 114. The P chip 104 makes use of lines 116 to enable data to be written to the cache memory 106. Further, lines 118 provide the data and error correction codes (ECC) from the cache memory 106 to the processing chip 104.

The cache memory 106 is shown in this example composed of random access memories (RAMs). The cache 106 further receives data and ECC information from the CF chip 100 over lines 120. The operation of a cache memory 106 along with its associated cache tag store 110 is well known to those skilled in the art and will not therefore be discussed.

The X chip 102 provides an interface to the XJA bus adaptor 25 (Fig. 1). The X chip 102 provides outputs to the CF chip 100 and receives inputs therefrom. Further,

the X chip 102 interfaces with a console support block 108 for interfacing with a console terminal 26 (Fig. 1).

The CF chip 100 provides outputs to the cache tag store 110 including new tag entry and write enable information. Further, the CF chip 100 receives address and data operands, along with parity information, from the processing chip 104. The CF chip 100 couples the CPU with the E BUS TA and E BUS FA as indicated by bus lines 17 and 21, respectively.

Interlock Queueing

Referring back to Fig. 1, it is often desirable for some device such as one of the CPU modules 0-3 or an I/O unit or device through the XJA adaptor to gain exclusive access to a particular unit or location of the memories 31a-31n. For example, one of the CPU modules 11-14 may desire to lock a block of data, a longword, a byte, etc., in one of the memory modules 31a-31n. To do so, the device generates a lock request in the form of a READ LOCK command. This command will lock the particular memory location allowing the resource exclusive access thereto. Once the location is locked no other CPU's lock request to that location will be allowed. Once the resource has completed its operations on the location, a WRITE UNLOCK command is generated to free the memory location. The precise operation of these commands is described below with respect to Figure 4.

In the example of Fig. 1, there are four E BUS nodes, corresponding to CPU modules 0-3, from which lock requests can originate. Still further, as seen from Fig. 3, each node on the E BUS, in this case the CPUs, also includes more than one resource which may generate lock requests, for example, the P chip 104 or the X chip 102. In this instance, the CF chip 100 functions to insure that neither the processing chip's 104 or X chip's 102 lock requests are locked out from being serviced. Similarly, the central unit 15 must insure that no lock out occurs for any of the nodes on the E BUS.

Referring to Fig. 2, the central unit 15 makes use of lock block 71 which includes lock logic to be described below in Fig. 4, to insure that no node is ever locked out from having its lock request satisfied. Further, the CF chip 100 in the CPU shown in Fig. 3 includes lock logic 125 for insuring that neither the CPU nor the X chip are ever locked out from having their lock requests satisfied. In this manner as will be fully described below, a hierarchy of lock out avoidance mechanisms, i.e., lock logic 71 and lock logic 125, are utilized to insure that every lock request is eventually serviced.

Referring to Fig. 2, lockout avoidance is performed at the highest level by the lock logic 71 in the central unit 15. The use of the central unit 15 to keep track of lock locations and to avoid lockouts provides an advantage over conventional multi-drop buses wherein each CPU or memory coupled to the bus must keep track of the locked data. Because the central unit 15 receives all lock requests and maintains lock registers, it provides the opportunity to implement centralized queueing mechanisms to avoid the lockouts.

Fig. 4 describes in greater detail the lock logic 71. The lock logic 71 includes a predetermined number of lock registers 80a-80n, in the example shown "n" equals eight, but it is understood to depend upon the requirements of the system. Each of the lock registers 80a-80n stores the address of a particular memory location to be locked. For this example having eight lock registers, up to eight memory locations can be locked at any given time. The outputs from each of the register blocks 80a-80n are the result of a compare performed between each of the lock registers and the address of an incoming lock request on bus 36. The output of OR gate 82 provides either a LOCK HIT or LOCK MISS signal. The LOCK HIT signal is provided to further logic 84 which outputs a LOCK REFUSED signal to the resource check 67 in the arbiter 51 (Fig. 2). The LOCK MISS signal is provided as an input to a write control 86.

Each lock register 80a-80n includes a register 88 and a comparator 90. The register 88 stores an address of the memory location desired to be locked along with an indication of its validity, i.e., locked or unlocked, as shown in Fig. 4a. The register 88 is composed of several fields including a lock register valid bit field, node source ID, and byte addresses. The register 88 is enabled via commands from the write control block 86.

The command and address received from the schedule logic 66 (Fig. 2) over bus 36 is provided in parallel to the comparators 90 in each of the lock registers 80a-80n. The command and address are further provided to delay logic 92 which has an output coupled in parallel to the registers 88 in each of the lock registers 80a-80n. The output from the registers 88 are provided as another input to the comparators 90 in each of the lock registers. The output from the comparators 90 in each of the lock registers is then provided to the OR gate 82 mentioned above.

A lock queue, shown generally by dotted line 130, is composed of several state devices 94a-94d chained together. The number of entries in the lock queue (in this example four) correspond to the number of nodes. An example of one of the state devices 94a-94d is shown in greater detail in Fig. 4b functioning as a lock queue register entry including a lock queue register valid bit and node source ID bits. Coupled with each state device 94a-94d is an associated multiplexer 96a-96d. The registers 94a-94d and multiplexers 96a-96d are coupled together to form the lock queue 130 which functions as will be described below. A comparator 98a-98d is associated with each individual state device and multiplexer 94, 96 and receives an input therefrom. Further, the comparator 98 receives an input from the command address bus 36. Each of the comparators 98a-98d provides an output to OR gate 99. The OR gate 99 provides inputs to OR gate 84 and to write control logic 86. The write control logic 86 also receives inputs from each comparator 90 in the lock register blocks 80a-80n and

each comparator 98a-98d associated with the lock queue 130.

The write control logic 86 provides a LOCK REGISTER WRITE CONTROL signal to each of the registers 88 in the lock registers 80a-80n. Further, the write control block 86 provides a LOCK QUEUE REGISTER WRITE CONTROL signal to enable the multiplexers 96a-96d associated with state devices 94a-94d to operate in accordance with the lock queue 130 design. Also, a REFUSAL COUNTER INCREMENT CONTROL signal is provided from write control 86 to a plurality of refusal counters 131-136. Each refusal counter is associated with a node, such as CPU modules 0-3 (Fig. 1) in the system. The refusal counters 131-136 are incremented by signals from the write control 86. Once a particular refusal counter reaches a predetermined limit, an END OF COUNT signal is provided to the write control 86. The predetermined limit in the refusal counter is optimized for the best performance for lock requests in the system. The write control 86 further receives from the arbiter 51 (Fig. 2) a LOCK CLEAR and LOCK SET signal.

In operation, assuming all of the lock registers 80a-80n are available for use, a READ LOCK command and an associated memory address to be locked are provided in parallel to the comparators 90 in each of the lock registers 80a-80n. Each of the respective comparators 90 compares the address with the address previously stored in its associated register 88. In this example, because each of the lock registers is initially empty, the comparators 90 do not generate any matches as an output to the OR gate 82. Therefore, the output from the OR gate 82 provides a LOCK MISS signal to the write control 86. The write control 86 then provides a LOCK REGISTER WRITE CONTROL signal to registers 88 to enable a particular one of the registers 88 in the lock registers 80a-80n and to set the valid bit. The particular block which is enabled by the LOCK REGISTER WRITE CONTROL signal is determined by a predefined protocol in the write control 86. By enabling the register 88 with the signal, the READ LOCK command stores its associated address in the register 88 from line 89 after having been provided as an input thereto from bus 36 through the delay logic 92. Further the write control 86 sets the valid bit in register 88 to indicate the memory location is locked. The delay logic 92 functions to sufficiently delay the command and address so as to arrive at the register 88 once the write control 86 generates its LOCK REGISTER WRITE CONTROL signal, enabling storage at a predetermined location. In this manner, up to eight lock addresses can be stored in the lock registers 80a-80n.

Associated with each READ LOCK command is a WRITE UNLOCK command and associated address. The WRITE UNLOCK command is necessary to unlock the particular memory location once the resource has finished its use. Referring back to Fig. 4, WRITEUNLOCK commands are never refused by the lock logic 71. A WRITE UNLOCK command and associated address on bus 36 provides a LOCK MISS output to the write control 86 which provides a LOCK REGISTER WRITE CONTROL signal to clear the valid bit in the particular register 88 in the lock register block 80a-80n corresponding to the WRITE UNLOCK command.

Once all of the lock registers are filled with valid addresses of locked locations, as indicated by the lock register valid bit in register 88 (Fig. 4a), then any subsequent lock requests will be refused. Similarly, if a lock request is generated for a particular memory location which is already locked, then that lock request will also be refused. For example, assuming a first READ LOCK command was allowed for address 00 in the memory, then one of the registers 88 will contain address 00 and its lock register valid bit will be set. If a subsequent READ LOCK command also desires to access address 00, then the one comparator 90 associated with the register 88 storing that particular address will generate a match signal to OR gate 82. The OR gate 82 thus provides a LOCK HIT signal to OR gate 84 which in turn outputs a LOCK REFUSED signal to the resource check 67 in the arbiter.

Referring back to Fig. 2, the LOCK REFUSED signal is provided as an input to the resource check 67 which enables multiplexer 37b to provide a retry signal on the particular E BUS 21-24 from which the READ LOCK command was generated.

Referring back to Fig. 4, when a match is generated from one of the comparators 90 in one of the lock registers 80a-80n, a signal is sent over line 200 to the write control 86. The match indicates that the memory location is currently locked. The write control 86 then provides a REFUSAL COUNTER INCREMENT CONTROL signal. The REFUSAL COUNTER INCREMENT CONTROL signal enables the particular refusal counter 130-136 associated with the node generating the READ LOCK command which was refused. This signal increments the counter.

Similarly, when all of the lock registers 80a-80n are filled with a valid address and not the address for which the lock is requested, then the LOCK MISS signal provided from OR gate 82 to the write control 86 will enable the REFUSAL COUNTER INCREMENT CONTROL signal. This is possible because the write control 86 internally tracks the number of valid lock registers and therefore knows when the lock registers 80a-80n are filled.

Once a refusal counter 131-136 has been incremented a predetermined number of times, the END OF COUNT signal is provided to the write control 86. This generates a LOCK QUEUE REGISTER WRITE CONTROL signal from the write control 86 to the lock queue 130. The node source ID and the lock queue register valid bit (Fig. 4b) are then loaded and set, respectively, into the top of the queue 94d by enabling the multiplexer 96d for the particular READ LOCK command and associated address from bus 36. This lock request then be-

comes the head of the lock queue 130. Once the lock queue 130 contains a valid entry, any subsequent lock requests from other nodes will be automatically refused (regardless of whether the other conditions for refusing locks are satisfied) and will be stored in the queue 130.

The LOCK QUEUE REGISTER WRITE CONTROL signal can implement several functions. These functions include a Hold signal, Update signal and Push To Top of Queue signal. These signals are described below with respect to the queue 130 operation.

The LOCK QUEUE REGISTER WRITE CONTROL signal is next input to multiplexer 96c which enables the second lock request to be stored in location 96c in the lock queue 130. If another lock request is generated from a node already stored in the lock queue 130, that request will then generate the LOCK QUEUE REGISTER WRITE CONTROL hold signal that enables the middle line of the multiplexers currently storing the same entry 94. This new request then is stored in the same queue position.

Each entry in the lock queue 130 is provided as an input to its associated comparator 98a-98d. Further, the information on bus 36 is also provided to each of the comparators 98a-98d. The output from the comparators 98a-98d are provided as parallel inputs to OR gate 99 which provides an output both to OR gate 84 (QUEUE HIT) and to write control 86.

The lock queue 130, as shown in Fig. 4, has as many entries as there are nodes coupled to the central unit 15. As used in the present example, the four entry lock queue, i.e., entries 94a-94d, is required because of the four possible E BUS nodes from which lock requests may originate. Again, as shown in Fig. 4b, each entry in the queue 130 is composed of two parts: a valid bit, indicating whether the entry is valid, and a slot ID which points to one of the E BUS CPU nodes.

Every time the head entry 94d of the lock queue 130 is serviced, i.e., its lock request is satisfied, the lock queue is pushed by the LOCK QUEUE REGISTER WRITE CONTROL push to top of queue signal enabling the lower input to the multiplexers 96a-96d. This has the effect of pushing the entries upward in the queue to create a new head of queue and making available the bottom entry 94a. For this reason, the lock queue 130 need only be of a depth equal to the number of nodes on the E BUS which can generate lock requests.

An example of the lockout avoidance mechanism operation will now be given. Lock requests are generated from the various nodes (Fig. 1) in the form of READ LOCK commands and an associated address location and are received by the central unit 15 through their respective port logics 49a. The request passes through the scheduling logic 66 and into the lock logic 71 (Fig. 2) on bus 36. As long as the lock request is not refused by the lock logic 71, the request does not enter the lock queue 130 (Fig. 4). However, assuming the lock request from node 0, i.e., CPU 0, is refused because the particular location in memory has already been locked or because no free lock registers 80a-80n are available for allocation to the lock request, then the write control 86 increments the refusal counter 131-136 for the particular node, in this case CPU 0. After a predetermined period of time, CPU 0 again retries its lock request. If the lock request from CPU 0 is again refused, its respective refusal counter is incremented once more. Once a lock request has been refused a predetermined consecutive number of times while the lock queue 130 remains empty, the write control 86 then places the identification for the particular node at the head of the lock queue 130. Only the node ID (Fig. 4b) is stored in the lock queue and not the address of the lock request that was refused.

Because the lock queue 130 is no longer empty, a subsequently arbitrated lock request from any other node will be refused and will also enter the lock queue 130 in the order in which it is serviced by the arbiter 51. Once a lock request from the head of the queue is serviced, the next waiting lock request in the lock queue 130 is pushed to the head of the queue via the LOCK QUEUE REGISTER WRITE CONTROL push to top of queue signal and the operation of the multiplexers 96a-96d.

This operation assures that whenever the lock queue 130 is empty, it is difficult to enter the lock queue 130 because the lock requests from a particular node must be refused a predetermined consecutive number of times. Therefore, the lock queue 130 is predominantly kept empty, assuming the refusal counter is set at a high enough value. However, once the lock queue 130 has an entry, all future lock requests are serviced strictly by their lock queue position, i.e., the head of the queue gets satisfied first, the second entry is satisfied next, etc. This operation continues until the lock queue 130 is again empty.

In summary, the lock logic generates a LOCK REFUSED signaling response to a READ LOCK command from a node in three instances: 1) the lock queue is not empty and the requesting node is not the head of the queue; 2) no free lock registers are available; or 3) the particular memory block desired is already locked.

Because the lock requests are unconditionally queued in the lock queue 130 once the lock queue 130 is no longer empty, the address of the refused lock request is not important. So long as each node guarantees that it will never lockout any of the multiple sources of lock requests it receives, the central unit 15 guarantees that every lock request, regardless of its address, has a fair and equal chance of entering the lock queue 130 and hence, eventually reaching the head of the lock queue 130 for servicing.

The above description guarantees that no lockout will occur at the highest level in the system. However, to guarantee that lockouts do not occur anywhere in the system, lockout avoidance mechanisms must be in place wherever multiple sources of lock requests are generated. Thus, while the central unit 15 avoids lockouts for each of the nodes, i.e., the CPUs coupled there-

to, via lock logic 71, a lockout avoidance mechanism is necessary in each node to avoid lockouts of other sources of lock requests.

Referring again to Fig. 3, the CPU block diagram includes the CF chip 100 which has a lock logic 125. This lock logic 125 prevents lockouts of either the processor chip's 104 or X chip's 102 lock requests. The lock logic 125 operates similarly to the lock logic 71 shown in Fig. 2. Because the lock queue 130 in the central unit 15 does not store addresses nor whether the lock request was from either the X chip or the processor chip 104, it is possible that either the processor chip or the X chip can lock the other out from ever having their lock request satisfied.

Fig. 5 illustrates the two entry queue used in the lock logic 125 of the CF chip 100. In this example, the queue 134 includes two entries, each of which includes two bits: 1) an entry valid bit; and 2) a source ID bit, e.g., X chip or processor chip.

In operation, as long as a lock request from the processor chip or X chip is not refused by the central unit 15, the queue 134 remains empty. However, once a lock request from, for example, the processor chip, is refused by the central unit 15, the lock request immediately enters the head of the queue 134. From this point onward, only the processor chip can have its lock requests forwarded onto the E BUS (and hence to the central unit 15) from the CF chip 100. If, for example, the X chip requests a lock, this request will enter the queue 134 behind the processor chip's entry. Further, the X chip's lock requests will internally be refused by the CF chip 100 and thus prevented from going onto the E BUS.

Once the head of the queue, in this example the processor chip entry, is eventually satisfied by the central unit 15, it is popped out of the queue 134 and the X chip's lock request becomes the new head of the queue 134.

The method used in the CF chip 100 for avoiding lockouts is as follows. If the queue 134 is empty, both the processor and X chip can send their respective lock requests to the central unit 15 via the E BUS 17-20. If a lock request is refused by the central unit 15, then the source of the lock request, i.e., processor or X chip, along with whether it is valid or invalid, is entered into the queue 134. It is important to note that if the source is already stored in the queue, it will maintain its queue position.

The protocol used in the CF chip to avoid lockouts is essentially similar to the lock queue 130 used in the central unit. However, in this instance there is no need to implement a refusal counter in the CF chip 100, because the frequency at which lock requests are generated from the X chip at the CPU level is not great enough to require the use of a refusal counter.

Claims

1. A method for avoiding lockout of lock requests for a shared resource, generated from a plurality of nodes coupled to a central location, the method comprising the steps of:

   incrementing a counter associated with a particular one of the nodes when a lock request from said particular node is refused in the central location;
   generating an end of count signal from said counter when the lock request is refused a predetermined number of times;
   storing the lock request as a head entry in a queue once the end of count signal in generated;
   subsequently storing any further lock requests from any other of the nodes at the tail of the queue as further entries in said queue once the queue contains a head entry; and
   operating said queue to the entries in the order in which they are placed in the queue.

2. A method according to claim 1, wherein the step of storing the lock request comprises generating a first signal to enable a first register in said queue to store the lock request.

3. A method according to claim 2, wherein the step of subsequently storing comprises generating a second signal to enable a next register in said queue to store the next lock request from a different node.

4. A method according to claim 3, wherein the step of operating the queue comprises:

   generating a third signal to push said queue once the head entry is serviced such that the next entry in the queue becomes the head entry; and
   continuing to generate the third signal each time the head entry is serviced until the queue is empty.

5. A lockout avoidance circuit for a plurality of nodes coupled to a central location and generating lock requests for a shared resource, comprising:

   a lock queue having a head and tail;
   means for counting lock requests that have been refused from at least one of those nodes;
   means for enabling the queue to store a lock request from the node at the head of the queue after a predetermined number of lock requests from the node have been refused, and storing all subsequent lock requests from any other of the nodes at the tail of the queue in the order

in the order in which they are requested; and
means for enabling operating the queue to advance the stored lock requests from the tail end of the queue toward the head of the queue each time the lock request at the head of the queue is serviced.

6. A circuit according to claim 5,including a control circuit coupled to the lock queue, wherein the lock queue includes a plurality of queue registers each having a storage device and wherein the plurality of queue registers provide either a lock refused or lock miss signal in response to a lock request; and
the lock miss signal being forwarded to the control circuit to activate the first signal for storing the lock request.

7. A circuit according to claim 5 wherein said lock queue includes a plurality of queue registers and further comprises:

a plurality of multiplexers, one of the multiplexers being associated with each one of the queue registers, said multiplexers receiving a second signal for operating the lock queue.

8. A circuit according to claim 6 wherein the lock refused signal enables the control circuit to generate a third signal to the respective counter associated with the node generating the lock request.

9. A circuit according to claim 6, wherein each queue register comprises:

a valid indication field; and
a node source ID field.

10. A circuit according to claim 9 wherein each storage device includes:

an address field;
a node source ID field; and
a valid indicator.

Patentansprüche

1. Verfahren zum Vermeiden der Aussperrung von Verriegelungsanforderungen für ein gemeinsam genutztes Betriebsmittel, die von mehreren Knoten erzeugt werden, die an eine zentrale Einrichtung gekoppelt sind, wobei das Verfahren die folgenden Schritte enthält:

Inkrementieren eines einem besonderen der Knoten zugeordneten Zählers, wenn eine Verriegelungsanforderung von diesem besonderen Knoten in der zentralen Einrichtung abgewiesen wird;
Erzeugen eines Zählendesignals vom Zähler, wenn die Verriegelungsanforderung in einer vorgegebenen Anzahl abgewiesen worden ist;
Speichern der Verriegelungsanforderung als einen Kopfeintrag in einer Warteschlange, sobald das Zählendesignal erzeugt worden ist;
anschließend Speichern irgendwelcher weiterer Verriegelungsanforderungen von irgendwelchen anderen der Knoten am hinteren Ende der Warteschlange als weitere Einträge in der Warteschlange, sobald die Warteschlange einen Kopfeintrag enthält; und
Bearbeiten der Einträge der Warteschlange in der Reihenfolge, in der sie in der Warteschlange gesetzt werden.

2. Verfahren nach Anspruch 1, bei dem der Schritt des Speicherns der Verriegelungsanforderung das Erzeugen eines ersten Signals enthält, mit dem ein erstes Register in der Warteschlange freigegeben wird, um die Verriegelungsanforderung zu speichern.

3. Verfahren nach Anspruch 2, bei dem der Schritt des nachfolgenden Speicherns das Erzeugen eines zweiten Signals enthält, mit dem ein nächstes Register in der Warteschlange freigegeben wird, um die nächste Verriegelungsanforderung von einem anderen Knoten zu speichern.

4. Verfahren nach Anspruch 3, bei dem der Schritt des Bearbeitens der Warteschlange enthält:

Erzeugen eines dritten Signals, um die Warteschlange zu verschieben, sobald der Kopfeintrag bedient worden ist, so daß der nächste Eintrag in der Warteschlange zum Kopfeintrag wird; und
fortgesetztes Erzeugen des dritten Signals jedesmal, wenn der Kopfeintrag bedient wird, bis die Warteschlange leer ist.

5. Aussperrungsvermeidungsschaltung für mehrere Knoten, die an eine zentrale Einrichtung gekoppelt sind und Verriegelungsanforderungen für ein gemeinsam genutztes Betriebsmittel erzeugen, mit:

einer Verriegelungswarteschlange mit einem vorderen Ende und einem hinteren Ende;
einer Einrichtung zum Zählen von abgewiesenen Verriegelungsanforderungen von wenigstens einem dieser Knoten;
einer Einrichtung zum Freigeben der Warteschlange, um eine Verriegelungsanforderung von dem Knoten am vorderen Ende der Warteschlange zu speichern, nachdem Verriegelungsanforderungen in einer vorgegebenen

Anzahl von dem Knoten abgewiesen worden ist, und zum Speichern aller nachfolgenden Verriegelungsanforderungen von irgendwelchen anderen der Knoten am hinteren Ende der Warteschlange in der Reihenfolge, in der sie angefordert werden; und
einer Einrichtung zum Freigeben der Bearbeitung der Warteschlange, um jedesmal, wenn die Verriegelungsanforderung am vorderen Ende der Warteschlange bedient worden ist, die gespeicherten Verriegelungsanforderungen vom hinteren Ende der Warteschlange zum vorderen Ende der Warteschlange vorzuschieben.

6. Schaltung nach Anspruch 5, die eine mit der Verriegelungswarteschlange gekoppelte Steuerschaltung enthält, wobei die Verriegelungswarteschlange mehrere Verriegelungsregister enthält, wovon jedes eine Speichervorrichtung besitzt, und wobei die mehreren Warteschlangenregister als Antwort auf eine Verriegelungsanforderung jeweils entweder ein Signal für abgewiesene Verriegelung oder ein Signal für fehlende Verriegelung erzeugen; und
das Signal für fehlende Verriegelung an die Steuerschaltung geschickt wird, um das erste Signal zum Speichern der Verriegelungsanforderung zu aktivieren.

7. Schaltung nach Anspruch 5, bei der die Warteschlange mehrere Warteschlangenregister enthält und die ferner enthält:
mehrere Multiplexer, wobei jedem der Warteschlangenregister einer der Multiplexer zugeordnet ist, wobei die Multiplexer ein zweites Signal zum Bearbeiten der Verriegelungswarteschlange empfangen.

8. Schaltung nach Anspruch 6, bei der das Signal für abgewiesene Verriegelung die Steuerschaltung freigibt, damit sie ein drittes Signal für den jeweiligen Zähler erzeugt, der dem die Verriegelungsanforderung erzeugenden Knoten zugeordnet ist.

9. Schaltung nach Anspruch 6, bei der jedes Warteschlangenregister enthält:

ein Gültigkeitsanzeigerfeld; und
ein Quellknoten-ID-Feld.

10. Schaltung nach Anspruch 9, bei der jede Speichervorrichtung enthält:

ein Adressenfeld;
ein Quellknoten-ID-Feld; und
einen Gültigkeitsanzeiger.

Revendications

1. Procédé pour éviter l'interdiction d'accès pour des demandes de verrouillage d'une ressource partagée, générés à partir d'une multiplicité de noeuds couplés à un emplacement central, le procédé comportant les étapes consistant:

à incrémenter un compteur associé à un noeud particulier parmi les noeuds lorsqu'une demande de verrouillage en provenance dudit noeud particulier est rejetée dans l'emplacement central;
à générer un signal de fin de comptage à partir dudit compteur lorsque ladite demande de verrouillage a été rejetée un nombre prédéterminé de fois;
à emmagasiner la demande de verrouillage en tant qu'entrée de tête dans une file d'attente une fois que le signal de fin de comptage a été généré;
à emmagasiner subséquemment toutes autres demandes de verrouillage émanant de n'importe lesquels des autres noeuds en queue de la file d'attente en tant que d'autres entrées de ladite file d'attente une fois que la file d'attente contient une entrée de tête; et
à commander ladite file d'attente par rapport aux entrées suivant l'ordre de leur placement dans la file d'attente.

2. Procédé selon la revendication 1, dans lequel l'étape de stockage de la demande de verrouillage consiste à générer un premier signal pour autoriser un premier registre de ladite file d'attente à emmagasiner la demande de verrouillage.

3. Procédé selon la revendication 2, dans lequel l'étape de stockage subséquent consiste à générer un second signal pour autoriser un registre suivant de ladite file d'attente à emmagasiner la demande de verrouillage suivante en provenance d'un autre noeud.

4. Procédé selon la revendication 3, dans lequel l'étape de commande de la file d'attente consiste:

à générer un troisième signal pour pousser ladite file d'attente une fois que l'entrée de tête a été prise en charge de sorte que l'entrée suivante de la file d'attente devienne l'entrée de tête; et
à continuer à générer le troisième signal chaque fois que l'entrée de tête est prise en charge, jusqu'à ce que la file d'attente soit vide.

5. Circuit pour éviter les interdictions d'accès prévu pour une multiplicité de noeuds couplés à un em-

placement central et générant des demandes de verrouillage d'une ressource partagée, comportant:

une file d'attente de verrouillage ayant une tête et une queue;
des moyens pour compter les demandes de verrouillage qui ont été rejetées émanant d'au moins un desdits noeuds;
des moyens pour autoriser la file d'attente à emmagasiner une demande de verrouillage en provenance du noeud à la tête de la file d'attente après qu'un nombre prédéterminé de demandes de verrouillage en provenance du noeud ont été rejetées, et pour emmagasiner toutes les demandes de verrouillage subséquentes en provenance de n'importe quel autre des noeuds en queue de la file d'attente suivant l'ordre dans lequel elles sont faites; et
des moyens pour permettre de commander la file d'attente de façon à avancer les demandes de verrouillage emmagasinées depuis l'extrémité queue de la file d'attente vers la tête de la file d'attente chaque fois que la demande de verrouillage se trouvant en tête de la file d'attente est prise en charge.

6. Circuit selon la revendication 5, comprenant un circuit de commande couplé à la file d'attente de verrouillage, dans lequel la file d'attente de verrouillage comprend une multiplicité de registres de file d'attente ayant chacun un dispositif de stockage, et dans lequel la multiplicité de registres de file d'attente fournissent soit un signal de refus de verrouillage, soit un signal de non correspondance de verrouillage en réponse à une demande de verrouillage; et

le signal de non correspondance de verrouillage est acheminé vers le circuit de commande pour activer le premier signal pour emmagasiner la demande de verrouillage.

7. Circuit selon la revendication 5, dans lequel ladite file d'attente de verrouillage comprend une multiplicité de files d'attente de registres et comporte, en outre:

une multiplicité de multiplexeurs, l'un des multiplexeurs étant associé à chacun des registres de file d'attente, lesdits multiplexeurs recevant un second signal pour commander la file d'attente de verrouillage.

8. Circuit selon la revendication 6, dans lequel le signal de refus de verrouillage autorise le circuit de commande à générer un troisième signal à destination du compteur respectif associé au noeud générant la demande de verrouillage.

9. Circuit selon la revendication 6, dans lequel chaque registre de file d'attente comprend:

une zone d'indication de validité; et
une zone d'identification de source de noeud.

10. Circuit selon la revendication 9, dans lequel chaque dispositif de stockage comprend:

une zone d'adresse:
une zone d'identification de source de noeud; et
un indicateur de validité.

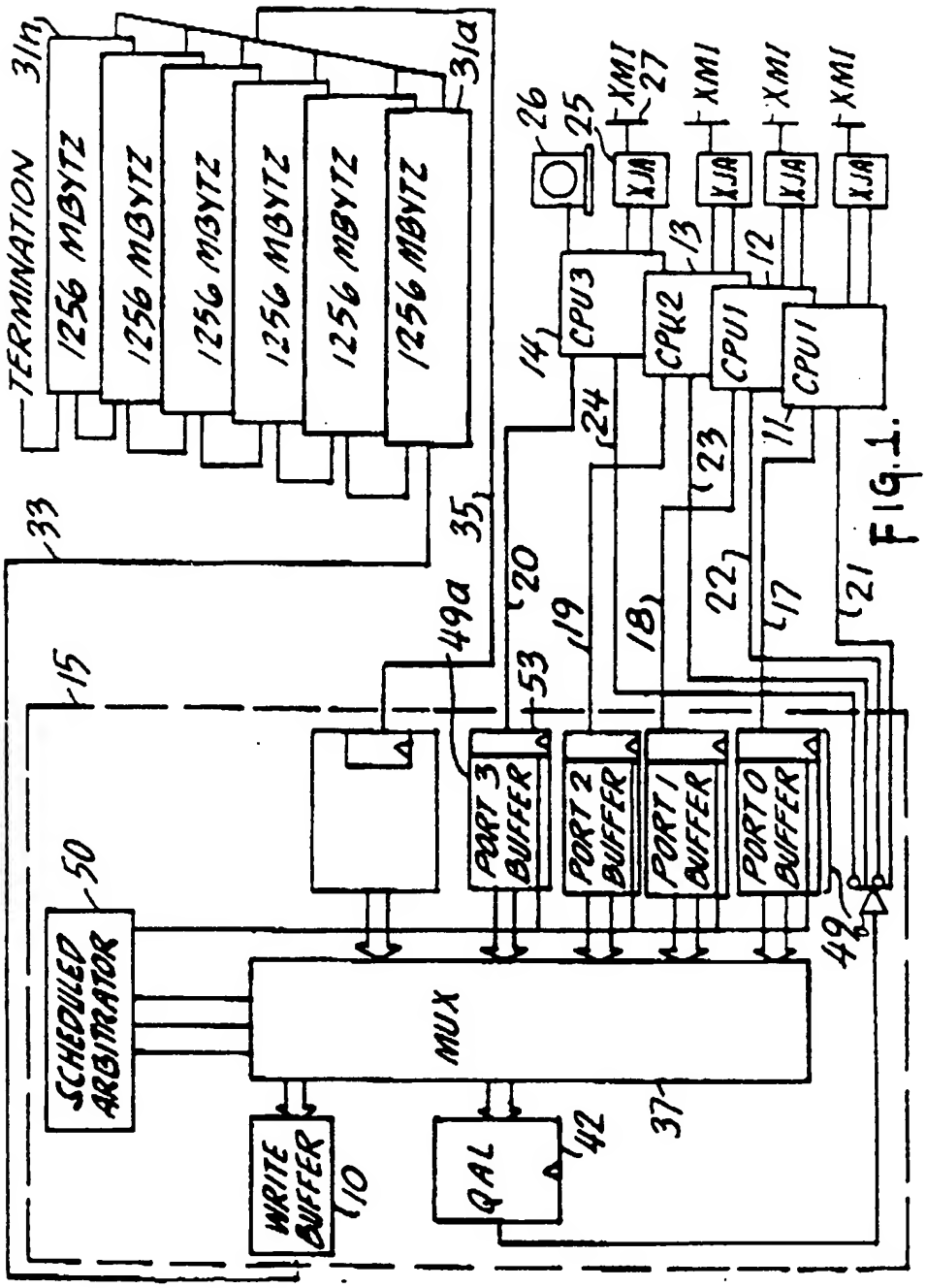
TERMINATION
31n
31a
1256 MBYTE
33
35
15
50
SCHEDULED ARBITRATOR
WRITE BUFFER
110
MUX
37
QAL
42
49a
49
53
PORT 3 BUFFER
PORT 2 BUFFER
PORT 1 BUFFER
PORT 0 BUFFER
20
19
18
22
17
21
24
23
11
14
CPU3
CPU2
CPU1
CPU1
13
12
26
25
XJA
XMI
27

FIG. 1.

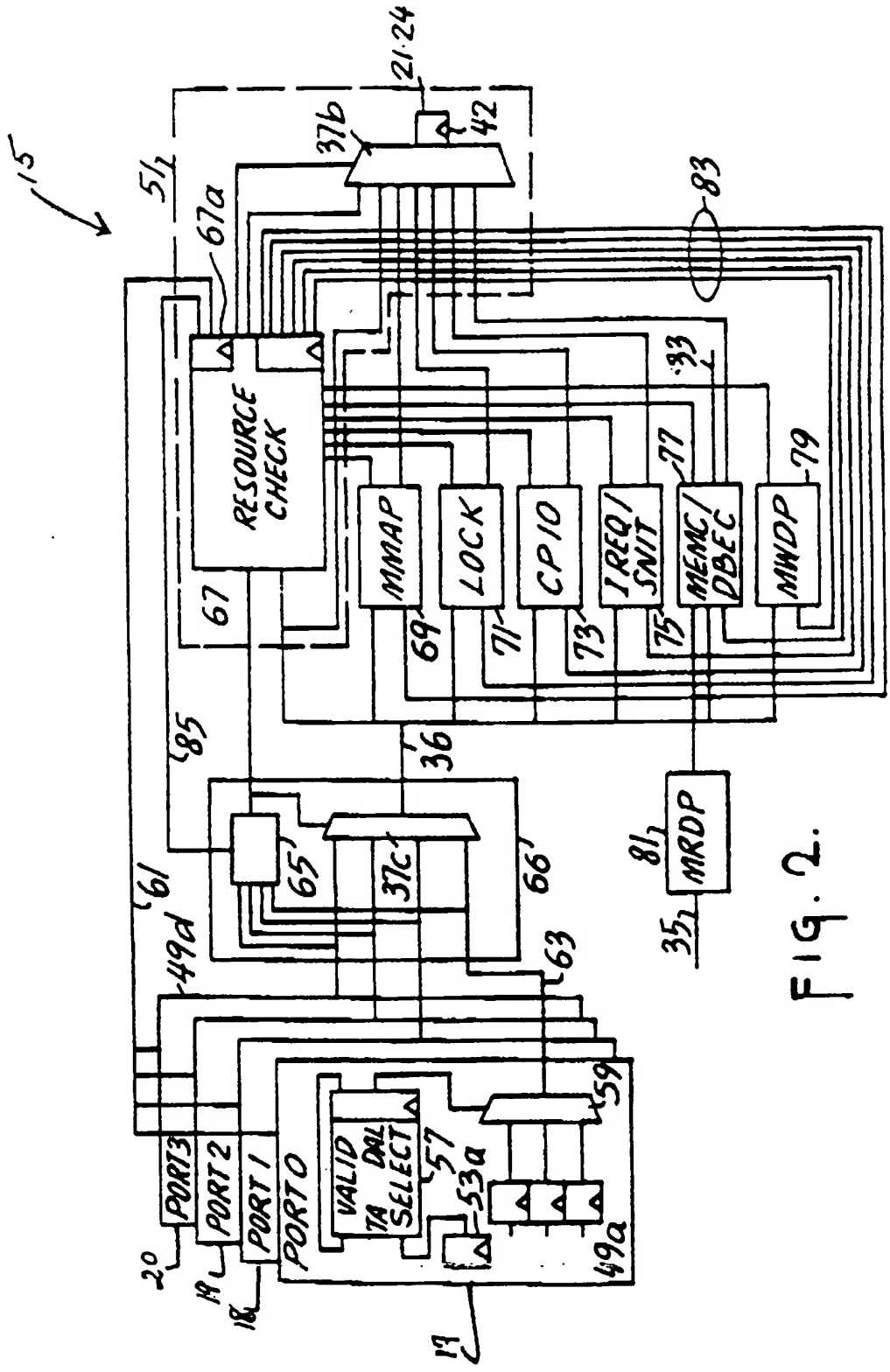
15
21·24
42
37b
51
67a
83
RESOURCE CHECK
67
MMAP
69
LOCK
71
CP10
73
IREQ/ SNIT
75
MEMC/ DBEC
77
MWDP
79
33
85
36
61
65
37c
66
63
49d
81
MRDP
35
PORT3
PORT2
PORT1
PORT0
VALID
TA
DAL
SELECT
57
53a
59
49a
20
18
16
17

FIG. 2.

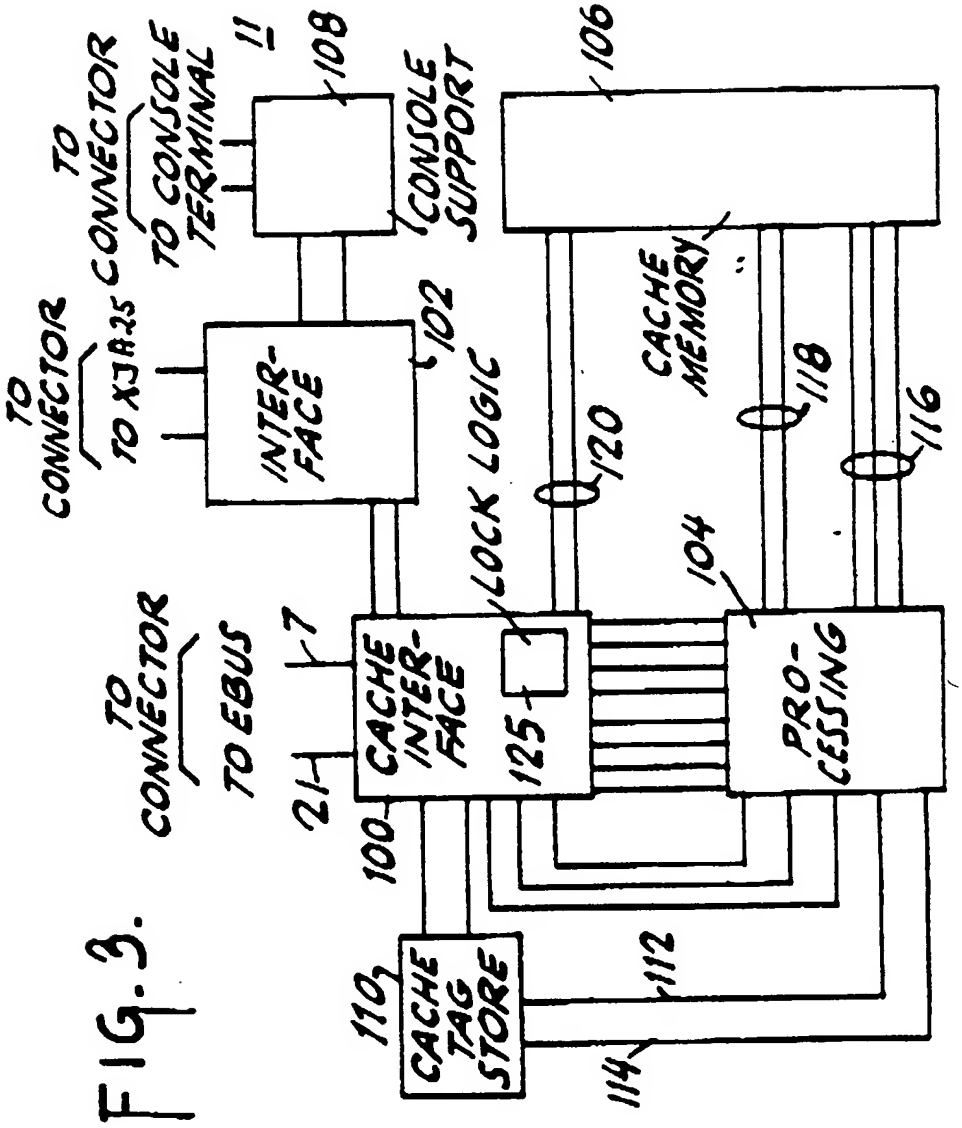
TO CONNECTOR
TO X3135
TO CONNECTOR
TO CONSOLE TERMINAL
11
108
CONSOLE SUPPORT
106
INTER-FACE
102
LOCK LOGIC
CACHE MEMORY
120
118
116
104
TO CONNECTOR
TO EBUS
7
21
CACHE INTER-FACE
125
PRO-CESSING
100
110
CACHE TAG STORE
112
114

FIG. 3.

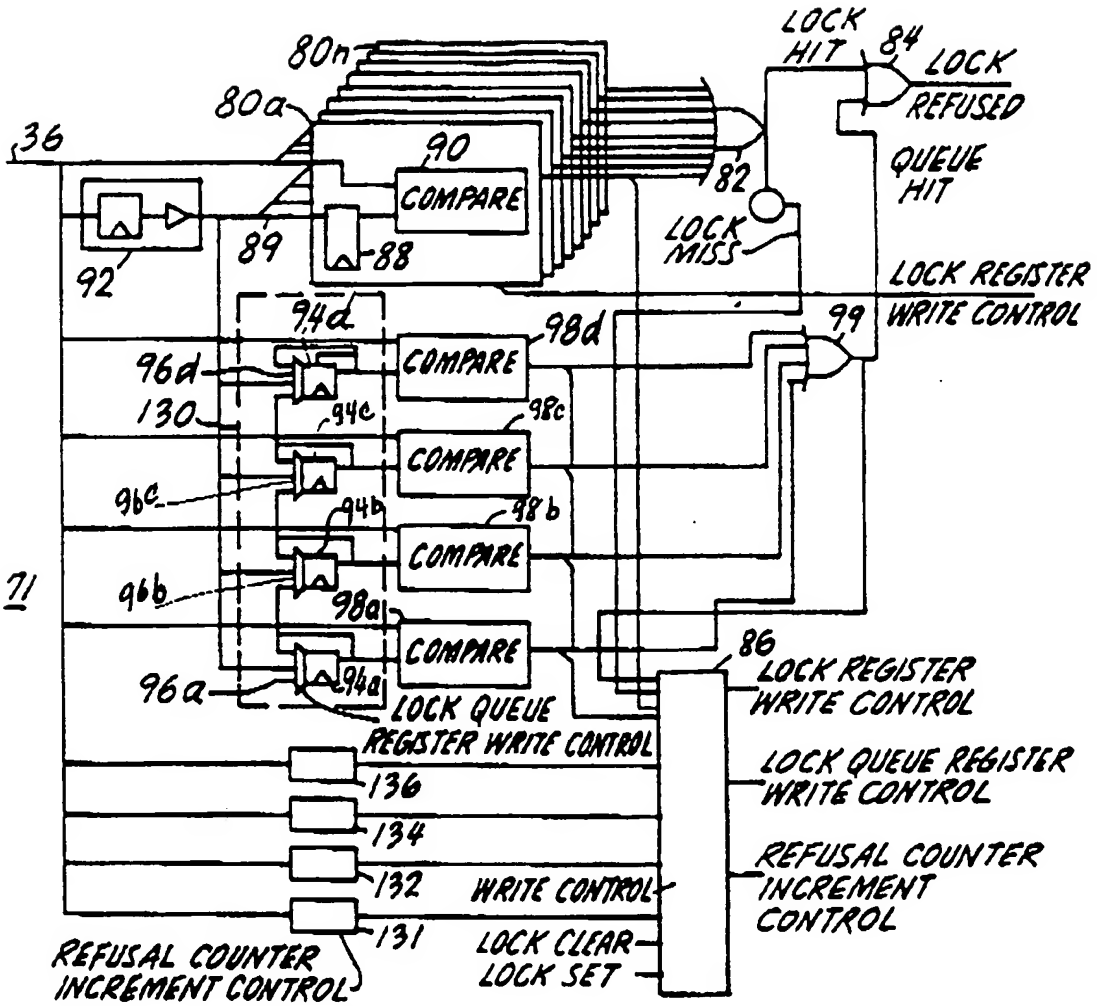
80n
80a
36
90
COMPARE
82
LOCK HIT
84
LOCK REFUSED
QUEUE HIT
LOCK MISS
89
88
92
LOCK REGISTER WRITE CONTROL
99
94d
98d
96d
130
94c
98c
96c
94b
98b
96b
71
98a
96a
94a
LOCK QUEUE REGISTER WRITE CONTROL
86
LOCK REGISTER WRITE CONTROL
LOCK QUEUE REGISTER WRITE CONTROL
136
134
132
131
WRITE CONTROL
REFUSAL COUNTER INCREMENT CONTROL
LOCK CLEAR
LOCK SET
REFUSAL COUNTER INCREMENT CONTROL

FIG. 4.

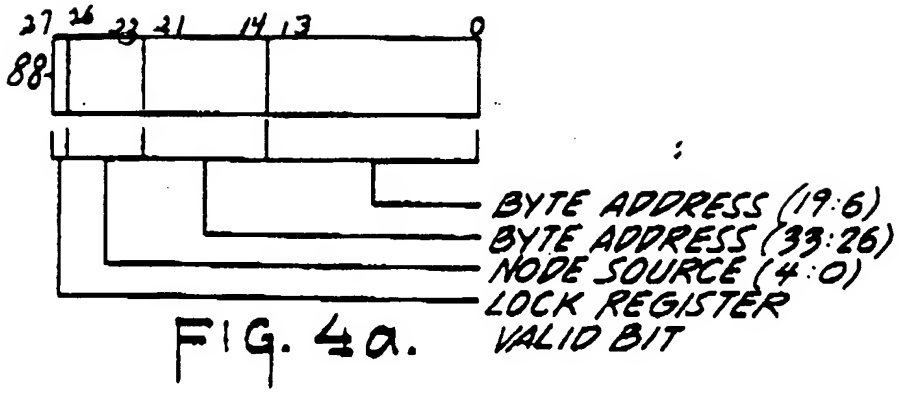

FIG. 4a.

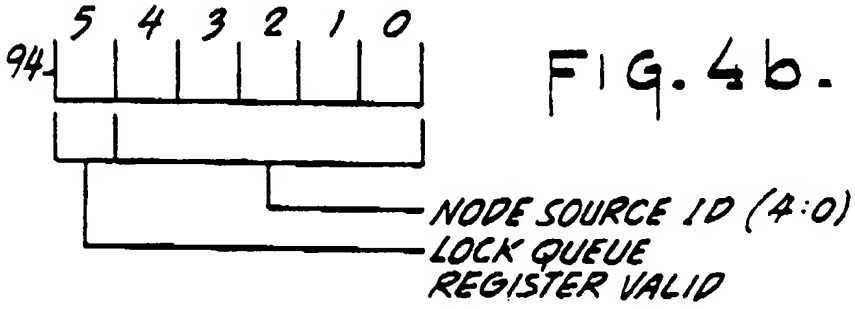

FIG. 4b.

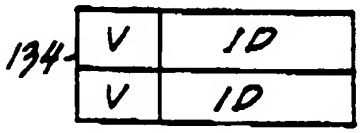

FIG. 5.